

Entity-Relationship Model

The logical database design process uses the entity-relationship model to represent the business scenario as a collection of entities and relationships. The entity-relationship diagram (ERD) documents the logical design and becomes a starting point for the physical design process, which defines the structure of the database. This lesson will discuss concepts related to the entity-relationship model.

The **Entity-Relationship** model is a tool that organizes and documents the logical design of a database. The entity-relationship model describes the data used by an application in terms of **entities, attributes and relationships**.

Entity

An entity is a person, place, thing or concept that is used in the business context of the application and for which data is collected. Synonyms for entity include "entity class" and "entity type". Most people use the term entity.

Consider an application that maintains customer orders for merchandise purchased from a hardware store. A sample order is shown below.

Order Number: 5		Date: October 10, 2025		
Nuts n Bolts Hardware 12538 - 77 Ave NW Edmonton, Alberta T6E 9K7 Phone/Text: 587-436-9812				
Customer Information Number: 100 Joan Adams 9825 - 77 Ave Edmonton, Alberta T5C 8E2 PH 780-488 - 8712				
Item	Description	Quantity	Price Per Unit	Extended Price
H-101	Widgets	10	1.00	10.00
H-229	Bolts	5	0.50	2.50
Subtotal				12.50
GST				0.88
Total				13.38

The entities involved in this information could include Order, Customer and Item.

Entities can be physical things that you can see (e.g. a customer) or conceptual things that do not physically exist (e.g. a course). It is important to distinguish between an **entity** and an **instance of an entity**. An instance of an entity is one specific occurrence of an entity. In the order example Customer is an entity. Joan Adams is an instance of the customer entity. An entity can have many instances (hopefully the hardware business is booming and there are many customers!).

Attributes

A piece of data that describes an entity is called an **attribute**. Synonyms for attribute include element, property and field. In the order example, the customer entity has the following attributes:

- Customer Number
- Last Name
- First Name
- Address
- Phone Number

The values for all attributes describing an entity make up one instance of the entity. For example, the following values describe one specific customer.

Customer Number	Last Name	First Name	Address	Phone
100	Adams	Joan	9825-77 Ave NW Edmonton AB T5C 8E2	780-488-8712

Types of Attributes

There are two types of attributes: **atomic** and **composite**. An atomic attribute has a value that is in its simplest form. Atomic attributes cannot be subdivided into two or more other meaningful attributes. "Customer Number" is an example of an atomic attribute. Generally speaking, storing data in atomic form adds flexibility to your application. A composite attribute is an attribute that can be decomposed into two or more atomic attributes. "Address" is an example of a composite attribute. Address can be broken down to street address, city, province, and postal code; all of which are atomic attributes.

Business rules play a role in deciding how to store data (atomic vs. composite). Consider the "Address" attribute. For a national company all addresses have the same format. Therefore, address is stored in its atomic form. For an international company addresses vary in format (Americans use a five-digit zip code while Canadians use a six-digit postal code). In this situation address would be stored in its composite form as a single attribute.

Attributes can be classified as **stored** or **derived**. A stored attribute has its value physically stored in the database. Order Number is an example of a stored attribute. A derived attribute can have its value calculated based on other data. Amount is an example of a derived attribute (amount can be calculated as Quantity * Price). Derived attributes need not be physically stored in the database. During the implementation phase of application development, a decision is made whether derived attributes should be stored in the database or not. If the decision is to physically store them, the database requires more space on the storage media. If the decision is not to physically store derived attributes, you save storage space, but you increase the resources used to execute the application. The application programs or the DBMS must now execute the instruction(s) to calculate the value of the derived attribute each time it is used. Either method can be used in a fully functional application.

Values an Attribute Can Assume

An attribute has a set of legitimate values. **These values must "make sense" for the business situation.** Using the customer order example, the value of the Customer Number attribute must be a number greater than zero. An attribute's legitimate set of values is defined in terms of the **data type** of the attribute and the **domain** of the attribute. The domain is the set of values that an attribute can assume. The set of legitimate values for the Customer Number attribute is defined as a data type of integer and a value that is greater than zero.

Special Case: The Null Value

Null is a special value that an attribute can assume. There are two fundamental ways to interpret the null value:

1. The value is unknown.

Consider creating an instance of the customer entity. If the customer does not know their postal code the null value can be stored as the postal code for the new record. The customer can supply their postal code at a later date. The record can then be updated to reflect the correct postal code.

2. This attribute does not apply for this particular instance.

Consider an employee entity. One attribute could be the name of the employee's spouse. The null value would be recorded in the spousal name attribute for all records that represent a single employee.

The null value must be used with caution, as it propagates through expressions. This means that any expression using an attribute that contains the null value will result in null.

Primary Key

Many tasks within an application involve using one specific instance of an entity. For example, changing the price of a specific item in inventory. To accomplish this, the application must have a way of uniquely identifying each instance of the item entity. The **primary key** allows the DBMS to uniquely identify each instance of an entity. The primary key is an attribute, or group of attributes, that has a unique value for each instance of an entity. In the Customer Order example, the Item Number attribute uniquely identifies each item in inventory and can act as the primary key for the item entity.

Sometimes an entity does not have a naturally occurring attribute that can act as the primary key. In this case, the database designer arbitrarily makes up an attribute solely for the purpose of acting as the primary key. This is called a **technical key**. Customer Number is an example of a technical key. Customers do not have numbers when they walk in the door to the business. The Customer Number attribute exists so the DBMS can identify a specific instance of the customer entity.

In some cases, a combination of two or more attributes acts as the primary key for an entity. This is termed as a **concatenated key** or a **composite key**.

The bottom line is that each entity must have a primary key that enables the DBMS to identify each instance of the entity.

Relationships

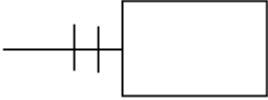
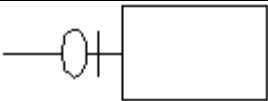
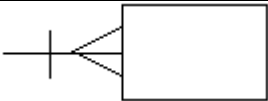
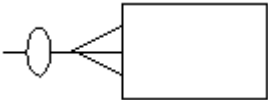
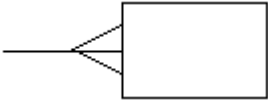
A relationship represents a business association between two entities. Relationships are based on the business rules of the organization. All relationships are bi-directional in that they can be interpreted from the point of view of each entity. In the customer order example, a relationship exists between the customer entity and the order entity. This reflects the fact that each time a customer makes a purchase an order is created. From the customer perspective, one customer can place many orders. From the order perspective, one and only one customer places an order.

Types of Relationships

Relationships are categorized based on how the instances of the entities are related to each other. The relating of instances between entities is called **cardinality**. The three major types of cardinalities are:

- One-to-One
 - Given an application that records parking stall assignments for employees, the business rule states: **Each employee is assigned one, and only one parking stall**. Therefore, one instance of the employee entity can be related to one instance of the parking stall entity, and one instance of the parking stall entity can be related to one instance of the employee entity.
- One-to-Many
 - Using the customer order example, the business rule states: **one customer can place many orders but one, and only one customer places an order**. Therefore, one instance of the customer entity can be related to many instances of the order entity, and one instance of the order entity can be related to one and only one instance of the customer entity.
- Many-to-Many
 - Using the customer order example, the business rule states: **many items can be purchased on one specific order, and one specific item can appear on many different orders**. Therefore, one instance of the order entity can be related to many instances of the item entity, and one instance of the item entity can be related to many instances of the order entity.

The **cardinality** defines the number of instances of an entity that can be related to a single instance of another entity. Common cardinalities include:

Cardinality	Symbol	Definition
To one		One instance of an entity is related to one instance of another entity. Drivers license: one instance of the person entity is related to one instance of the driver's license entity.
To zero or one		One instance of an entity is related to zero or one instance of another entity. Assigning parking stalls: one instance of the employee entity is related to one instance of the parking stall entity. Employees who do not drive to work do not participate in the relationship; therefore, not all instances of the employee entity would be related to a parking stall entity.
To one or many		One instance of an entity is related to one or many instances of another entity. Timecards for hourly employees: one instance of the employee entity is related to one or more instances of the timecard entity.
To zero, one or many		One instance of an entity is related to zero, one or many instances of another entity. Employees assigned to projects: one instance of the employee entity is related to zero, one or many instances of the project entity.
To many		One instance of an entity is related to many instances of another entity. Students enrolled in courses: one instance of the course entity is related to many instances of the student entity.

Defining Relationships

A **foreign key** is the mechanism that defines a relationship between entities. **A foreign key is a primary key of one entity that appears as an attribute of another entity.** In each relationship one entity has a foreign key and the other entity has the related primary key. The entity with the foreign key is referred to as the **child** entity in the relationship. The entity with the primary key is referred to as the **parent** entity in the relationship. The instance in the child entity is said to reference or refer to the related instance in the parent entity. Foreign keys are used to define all relationships between entities. The foreign key and related primary key must be the same data; however, the two attributes do not need to have the same name. The name of the foreign key attribute should be descriptive and "make sense" with the business scenario. In some cases, the foreign key attribute will have the same name as the related primary key attribute in the parent entity; however, this is not always the case.

The One-To-One Relationship

In the one-to-one relationship one instance from an entity is related to at most one instance of another entity.

Consider the following scenario. A company consists of multiple departments (e.g. Research, Marketing etc). Company policy states that every department has a manager. A manager can manage one and only one department within the company. This business rule translates to a one-to-one relationship between the employee entity and the department entity. This relationship can be defined by:

1. Adding the DepartmentID attribute (primary key of the Department entity) as a foreign key, named "ManagesDept", to the Employee entity.
2. Adding the EmployeeID attribute (primary key of the Employee entity) as a foreign key, named "Manager", to the Department entity.

Either method will work. Consider the first method, adding the DepartmentID to the Employee entity as a foreign key named Manages Dept.

Employee

EmployeeId (pk)	Last Name	First Name	Manages Dept (fk)
100	Smith	Jon	10
200	Adams	Jayne	
300	Everett	Beverly	5
400	Williams	Greg	
500	Stiles	Adam	

Department

DepartmentId (pk)	Department Name
5	Research
10	Marketing

This is not a good choice as many employees are not managers. This results in many instances in the Employee entity with the null value stored as the value for the Manages Dept attribute.

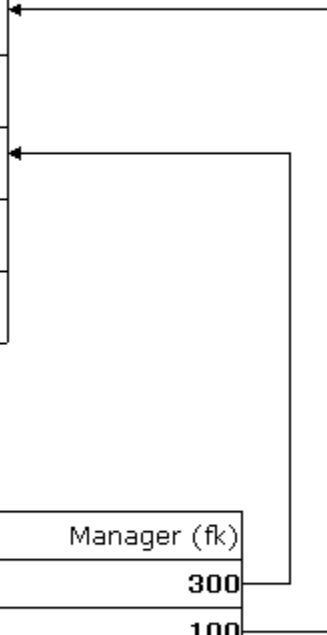
A better approach is to add the EmployeeID attribute to the Department entity as a foreign key named Manager.

Employee

EmployeeId (pk)	Last Name	First Name
100	Smith	Jon
200	Adams	Jayne
300	Everett	Beverly
400	Williams	Greg
500	Stiles	Adam

Department

DepartmentId (pk)	Department Name	Manager (fk)
5	Research	300
10	Marketing	100



Every department has a manager so an attribute that holds the null value for many instances is not introduced. In this implementation of the relationship the Employee entity acts as the parent while the Department entity acts as the child.

The One-To-Many Relationship

In the one-to-many relationship one instance from an entity is related to zero, one or many instances of another entity. Consider the following scenario. A retail sales company sells merchandise to customers. Each time a customer purchases goods an order is created. Over time a single customer can place many orders. Each individual order is made out to one customer. A customer must register with the company prior to making their first purchase. These business rules translate into a one-to-many relationship between the customer entity and the order entity (one customer can have many orders and a specific order is placed by one customer). The one-to-many relationship is defined by adding the primary key of the "one entity", as a foreign key, to the "many entity". The one entity (customer) is considered the parent entity, and the many entity (order) is considered the child entity.

Customer

Customer Number (pk)	Last Name	First Name	Address	City	Province	Postal Code	Phone
100	Jones	Grace	9825-77 Ave	Edmonton	Alberta	T5R 2E7	433-8126
110	Davis	Tyler	23 Apple Dr.	St. Albert	Alberta	T3Y 8B1	459-6194
120	Birney	Ann	13853 - 66 st	Edmonton	Alberta	T3R 2U8	466-8374

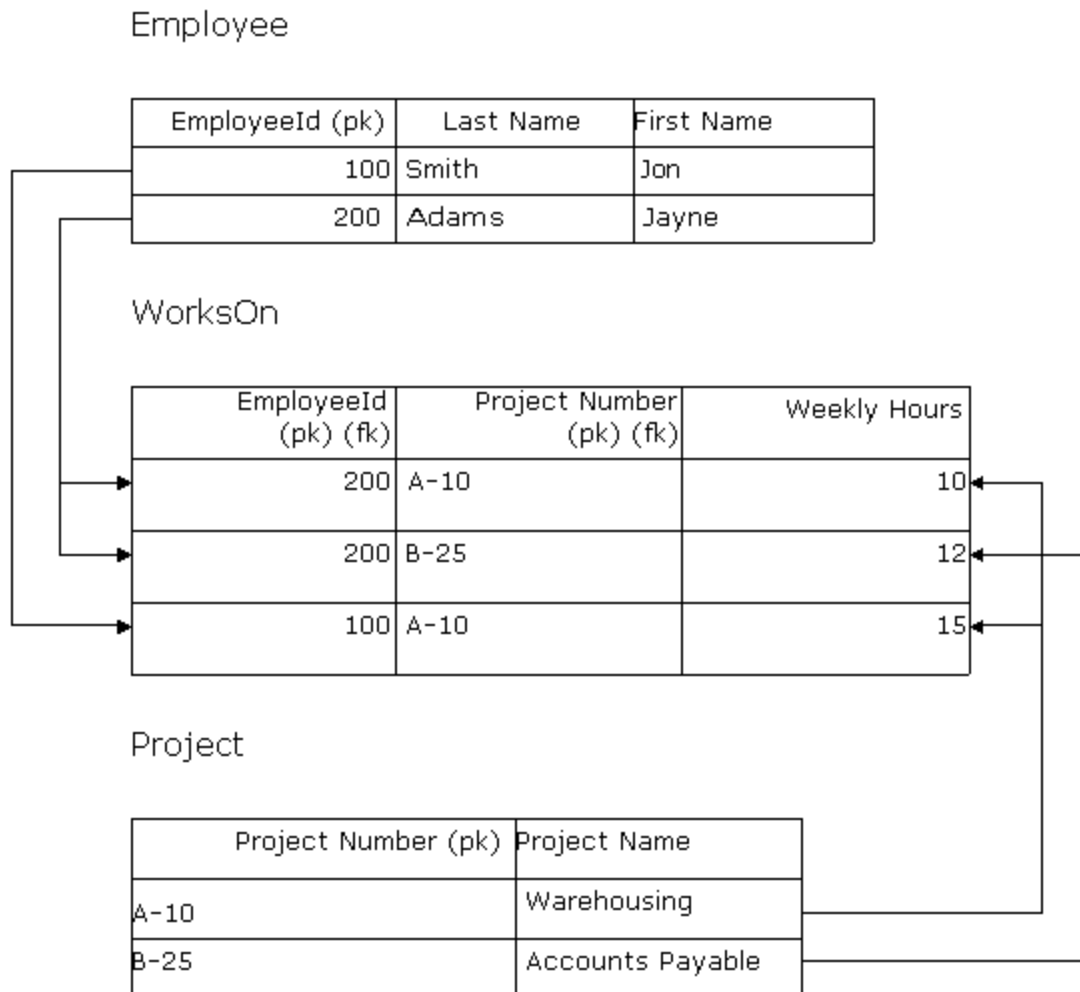
Order

Order Number (pk)	Date	Subtotal	GST	Total	Customer Number (fk)
5	Oct 10, 2000	12.50	0.88	13.38	120
10	Oct 22, 2000	10.00	0.70	10.70	110
15	Oct 24, 2000	20.00	1.40	21.40	110

The Many-To-Many Relationship

The relational database does not facilitate a many-to-many relationship between two entities. Hence, many-to-many relationships need to be reduced to one-to-many relationships. Creating a third entity referred to as an **associative entity** does this. The associative entity contains the primary key attributes of the two entities that share the many-to-many relationship as well as any other attributes that describe the association between the entities. Two one-to-many relationships are defined between the associative entity and the entities that share the many-to-many relationship.

Consider a consulting company. The company develops many projects simultaneously. Multiple employees make up a project team. One employee can be assigned to multiple projects simultaneously. These business rules translate into a many-to-many relationship between the Project entity and the Employee entity. The associative entity named **WorksOn** defines this relationship.



The WorksOn entity contains the following attributes:

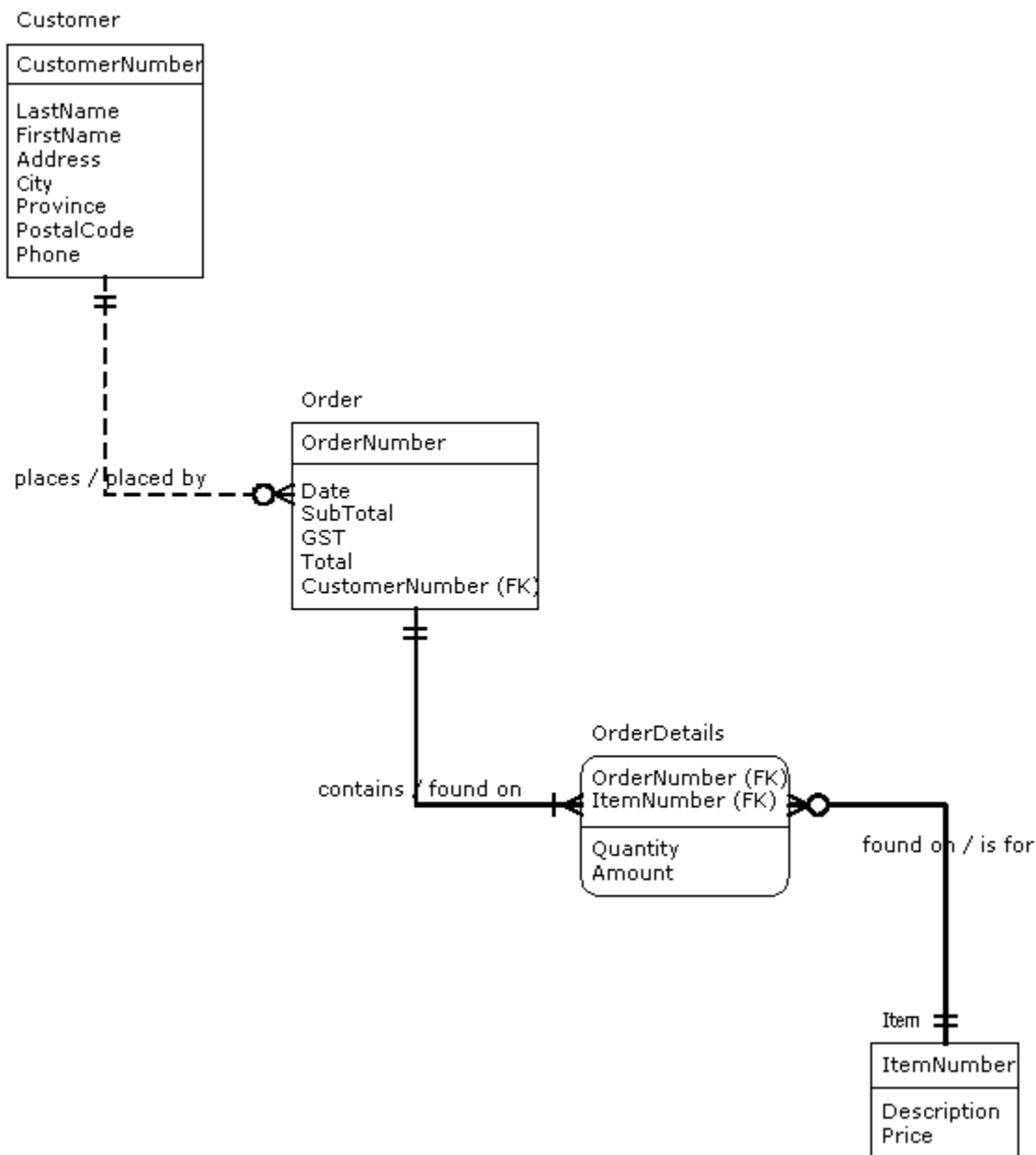
- EmployeeID (primary key of the Employee entity).
- Project Number (primary key of the Project entity).
- Weekly Hours (the hours per week the employee works on the project).

The primary key of the associative entity is always made up of two or more attributes. The primary keys from the entities that share the many-to-many relationship form the primary key for the associative entity. In this example, EmployeeID and ProjectNumber make up the primary key for the WorksOn entity.

A one-to-many relationship is defined between the Employee entity (parent) and the WorksOn entity (child). A one-to-many relationship is defined between the Project entity (parent) and the WorksOn entity (child). Note that a single attribute (e.g. EmployeeID in the WorksOn entity) can act as both a primary key and a foreign key within the same entity.

The Entity-Relationship Diagram (ERD)

The IT industry has developed several notations that can be used to construct an ERD (e.g. Chen, Martin, IDEF1X). Each notation has its own set of symbols for representing entities, attributes, keys and relationships. This course will use the IDEF1X notation. An ERD for the Customer Order application is shown below.



A box represents an entity. If the box has square corners, it is a base entity. If the box has rounded corners, it is an associative entity. The name of the entity appears above the box. All attributes describing the entity are listed inside the box. Primary key attributes are listed first and appear above the line. Attributes that do not act as the primary key for the entity appear below the line. Any attribute that acts as a foreign key are suffixed with "FK".

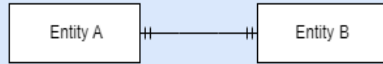
A line connecting two boxes represents a relationship between the two entities. The line has cardinality symbols that denote the type of relationship. A dashed line represents a **non-identifying** relationship. This means that the foreign key, in the child entity, does not act as part of the primary key for the entity. The relationship between the Customer entity and the Order entity is an example of a non-identifying relationship. A solid line represents an **identifying** relationship. This means that the foreign key does act as part of the primary key in the child entity. The relationship between the Order entity and the OrderDetails entity is an example of an identifying relationship. Relationships are labeled with a **verb phrase**. This allows the relationship to be read much like a sentence. The entities are the nouns, and the relationship label is the verb. Relationships can be read in two directions:

- From parent to child.
- From child to parent.

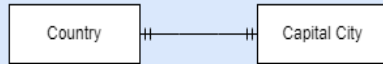
For the relationship between the Customer entity and the Order entity, Customer is the parent and Order is the child. Reading from parent to child: Customer places Order. Reading from child to parent: Order placed by Customer.

1:1 relationships

mandatory 1 : mandatory 1



each record in entity A maps to exactly one record in entity B.
each record in entity B maps to exactly one record in entity A.



e.g. each country has exactly one capital city.
each capital city belongs to exactly one country.

mandatory 1 : optional 1



each record in entity A maps to zero or one record in entity B.
each record in entity B maps to exactly one record in entity A.



e.g. each student has zero or one student visas.
each student visa belongs to exactly one student.

optional 1 : optional 1



each record in entity A maps to zero or one record in entity B.
each record in entity B maps to zero or one record in entity A.



e.g. each mentor has zero or one mentees.
each mentee has zero or one mentors.

1:many relationships

mandatory 1 : mandatory many



each record in entity A maps to at least one record in entity B.
each record in entity B maps to exactly one record in entity A.



e.g. each manager has at least one employee.
each employee has exactly one manager.

mandatory 1 : optional many

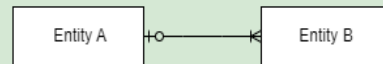


each record in entity A maps to zero, one, or many records in entity B.
each record in entity B maps to exactly one record in entity A.

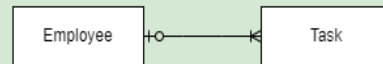


e.g. each student has made zero, one, or many payments.
each payment is for exactly one student.

optional 1 : mandatory many

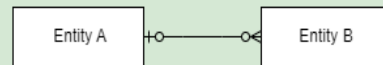


each record in entity A maps to at least one record in entity B.
each record in entity B maps to zero or one records in entity A.

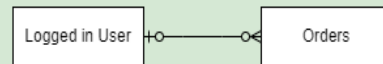


e.g. each employee is responsible for at least one task.
each task is assigned to zero or one employees (e.g. if the task has not yet been assigned).

optional 1 : optional many



each record in entity A maps to zero, one, or many records in entity B.
each record in entity B maps to zero or one records in entity A.



e.g. each logged in user has made zero, one, or many orders.
each order is for zero or one logged in users (e.g. if someone checked out as a "guest").